**2FA (Two-Factor Authentication) Demonstration Project**

**Subtitle**: Implementation and Documentation  
**Author**: Aditya  
**Date**: [Current Date]

**Table of Contents**

1. [Introduction](#Introduction)
2. [Project Overview](#Project-Overview)
3. [Technical Details](#Technical-Details)
   * [Libraries and Dependencies](#Libraries-and-Dependencies)
   * [Code Explanation](#Code-Explanation)
     + [Encrypting and Decrypting the Secret](#Encrypting-and-Decrypting-the-Secret)
     + [Generating the Secret Key](#Generating-the-Secret-Key)
     + [Generating the QR Code](#Generating-the-QR-Code)
     + [Validating OTPs](#Validating-OTPs)
     + [Main Program Flow](#Main-Program-Flow)
4. [Usage Instructions](#Usage-Instructions)
5. [Conclusion](#Conclusion)
6. [References](#References)

**Introduction**

Two-Factor Authentication (2FA) is a security process in which the user provides two different authentication factors to verify themselves. This project demonstrates a basic implementation of 2FA using One-Time Passwords (OTPs) and QR codes. The goal is to showcase how OTP-based 2FA can be implemented and validated using Python.

**Project Overview**

**Objective**: The objective of this project is to demonstrate the functionality of 2FA by generating a QR code for OTP provisioning, validating user-provided OTPs, and handling failed attempts with a blocking mechanism.

**Features**:

* OTP Generation and Validation
* QR Code Generation
* Handling of Failed Attempts and Blocking
* Regeneration of QR Code and Secret Key

**Technical Details**

**Libraries and Dependencies**

The following libraries are used in this project:

* pyotp: For OTP generation and validation.
* qrcode: For generating QR codes.
* pillow (PIL): For image processing and adding text to the QR code.
* pycryptodome: For AES encryption and decryption.

**Code Explanation**

**Encrypting and Decrypting the Secret**

* **Encrypting the Secret**: Uses AES encryption in CBC mode to encrypt the base32 secret key. The encrypted secret is then base64-encoded for storage.

python

Copy code

def encrypt\_secret(secret: str) -> str:

cipher = AES.new(aes\_key, AES.MODE\_CBC, aes\_iv)

encrypted\_secret = cipher.encrypt(pad(secret.encode(), AES.block\_size))

return base64.b64encode(encrypted\_secret).decode()

* **Decrypting the Secret**: Decrypts the base64-encoded encrypted secret using AES decryption and returns the original base32 secret key.

python

Copy code

def decrypt\_secret(encrypted\_secret: str) -> str:

encrypted\_secret\_bytes = base64.b64decode(encrypted\_secret)

cipher = AES.new(aes\_key, AES.MODE\_CBC, aes\_iv)

decrypted\_secret = unpad(cipher.decrypt(encrypted\_secret\_bytes), AES.block\_size).decode()

return decrypted\_secret

**Generating the Secret Key**

* **Generate Secret Key**: Creates a new base32 secret key and encrypts it.

python

Copy code

def generate\_secret\_key() -> str:

secret = pyotp.random\_base32()

return encrypt\_secret(secret)

**Generating the QR Code**

* **Generate QR Code**: Generates a QR code from the OTP provisioning URI and adds a message instructing the user to scan the code with an authenticator app.

python

Copy code

def generate\_qr\_code(secret: str) -> Image:

decrypted\_secret = decrypt\_secret(secret)

otp\_uri = pyotp.totp.TOTP(decrypted\_secret).provisioning\_uri("user@example.com", issuer\_name="DemoApp")

qr = qrcode.make(otp\_uri)

# Add a message to the QR code

qr\_image = qr.convert("RGB")

draw = ImageDraw.Draw(qr\_image)

try:

font = ImageFont.truetype("arial.ttf", 20)

except IOError:

font = ImageFont.load\_default()

text = "Scan this QR code with a valid authenticator app."

text\_bbox = draw.textbbox((0, 0), text, font=font)

text\_width = text\_bbox[2] - text\_bbox[0]

text\_height = text\_bbox[3] - text\_bbox[1]

width, height = qr\_image.size

position = ((width - text\_width) // 2, height - text\_height - 10)

draw.text(position, text, fill="black", font=font)

return qr\_image

**Validating OTPs**

* **Validate OTP**: Checks the user-provided OTP against the encrypted secret. Handles failed attempts and implements a blocking mechanism if too many failed attempts are detected.

python

Copy code

def validate\_otp(secret: str, user\_otp: str, user\_id: str) -> bool:

current\_time = time.time()

# Check if the user is currently blocked

if failed\_attempts[user\_id] >= MAX\_ATTEMPTS:

if current\_time - last\_attempt\_time[user\_id] < BLOCK\_TIME:

print("Too many attempts. Try again later.")

return False

else:

failed\_attempts[user\_id] = 0

decrypted\_secret = decrypt\_secret(secret)

totp = pyotp.TOTP(decrypted\_secret)

if totp.verify(user\_otp):

failed\_attempts[user\_id] = 0

return True

else:

failed\_attempts[user\_id] += 1

last\_attempt\_time[user\_id] = current\_time

return False

**Main Program Flow**

* **Program Execution**: Generates the secret key and QR code, then prompts the user to enter an OTP. If the OTP is valid, it displays a success message and exits the program. If invalid, it continues to prompt for a correct OTP until successful or until the regeneration time has passed.

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

secret\_key = generate\_secret\_key()

qr\_code = generate\_qr\_code(secret\_key)

qr\_code.show()

user\_id = "user@example.com"

start\_time = time.time()

while True:

user\_otp = input("Enter the OTP: ")

if validate\_otp(secret\_key, user\_otp, user\_id):

print("Login successful!")

break

else:

print("Invalid OTP!")

# Check if regeneration time has passed (optional)

if time.time() - start\_time >= REGEN\_TIME:

print("Regenerating QR code and secret key for enhanced security...")

secret\_key = generate\_secret\_key()

qr\_code = generate\_qr\_code(secret\_key)

qr\_code.show()

start\_time = time.time() # Reset the timer

# Close the program after successful login

exit()

**Usage Instructions**

1. **Setup**:
   * Install the required libraries:

bash

Copy code

pip install pyotp qrcode pillow pycryptodome

1. **Running the Script**:
   * Execute the script:

bash

Copy code

python your\_script\_name.py

1. **Interaction**:
   * A QR code will be displayed. Scan this QR code with an authenticator app (e.g., Google Authenticator, Authy).
   * Enter the OTP generated by the authenticator app when prompted.
2. **Expected Behavior**:
   * If the OTP is valid, the script will display "Login successful!" and terminate.
   * If the OTP is invalid, the script will prompt for a new OTP until a valid one is entered or the regeneration time elapses.

**Conclusion**

This project successfully demonstrates a basic implementation of 2FA using OTPs and QR codes. The code showcases essential aspects of 2FA, including OTP generation, QR code creation, and OTP validation. Future improvements could include more advanced features such as user management and more sophisticated error handling.

**References**

* [pyotp Documentation](https://pyotp.readthedocs.io/)
* [qrcode Documentation](https://pypi.org/project/qrcode/)
* [Pillow Documentation](https://pillow.readthedocs.io/en/stable/)
* [pycryptodome Documentation](https://www.pycryptodome.org/)